



PROJETO  EDUCAÇÃO

DO FUTURO

MÓDULO 05

Node.js e Git

AULA 03



AULA 01

CONTEÚDOS

- Apresentação do Node.js
- Fundamentos do Node.js
- Arquitetura e Funcionamento
- Módulos e NPM
- Configurações do ambiente
- Leitura de Arquivos
- Exercícios



O que é Node.js?

- O Node.js é uma plataforma de execução para JavaScript.
- Trata-se de uma biblioteca utilizada por um interpretador durante a execução de programas.
- Ele é baseado na engine V8 da Google, escrita em C++.
- O Node.js permite o desenvolvimento de aplicações em JavaScript no lado do servidor.
- O código JavaScript é executado sobre C++ para oferecer alto desempenho.



O que é npm?

- O npm é um gerenciador de pacotes do Node;
- Vamos poder utilizar bibliotecas de terceiros, baixando elas pelo npm;
- E também executar determinados scripts no nosso programa;
- Dificilmente um software em Node.js não utiliza o npm;
- Os módulos externos ficam numa pasta chamada node_modules;
- Ela deve ser descartável, ou seja, a cada instalação do projeto baixamos todos os pacotes novamente;



Instalação Node Windows

- O download do Node.js é feito no site oficial: nodejs.org;
- Vamos baixar um arquivo .msi, que é o instalador;
- É interessante saber que o npm vem junto do Node;
- Após a instalação podemos testar o Node e o npm em um terminal, para validar a instalação;



Instalação Node Windows

- Verificação se a instalação correu corretamente.

```
Prompt de Comando
Microsoft Windows [versão 10.0.26100.2605]
(c) Microsoft Corporation. Todos os direitos reservados.

C:\Users\Leticia Lima>npm --version
10.9.0

C:\Users\Leticia Lima>node --version
v22.12.0

C:\Users\Leticia Lima>
```



Nosso primeiro programa

- Agora vamos criar algo mais sólido, um programa simples baseado em um arquivo;
- A extensão dos arquivos de Node serão .js
- Vamos executar o arquivo com o comando: `node <arquivo>`
- O código será interpretado e o programa executado;
- Vamos lá!



Nosso primeiro programa

- Exemplo do primeiro programa em Node.js

```
JS arquivo.js ×
curso_node-main > 1_INTRO > 1_primeiro_programa > JS arquivo.js
1 console.log('Hello World Node!')
2
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\Leticia Lima\Desktop\curso_node-main> node arquivo.js
```



O que são módulos

- Módulos são scripts reaproveitáveis, que utilizamos bastante programando em Node;
- Eles são divididos em três categorias;
- Internos: módulos que nós desenvolvemos;
- Core Modules: módulos que vem com o Node.js;
- Externos: módulos que instalamos via npm;



Módulos internos

- Os módulos internos são criados nas pastas do nosso projeto;
- Precisamos exportar o módulo;
- Podemos utilizar a instrução `module.exports`;
- E importar onde precisamos utilizar;
- Para importar vamos utilizar a instrução `require`;
- Vamos criar um módulo!



Módulos internos

- Exemplo do primeiro módulo
 - Crie um arquivo meu_modulo.js e escreva uma função como no exemplo, depois crie um novo arquivo onde seu aplicativo principal ficará e importe o módulo como no exemplo

```
JS meu_modulo.js X JS index.js
curso_node-main > 2_FUNDAMENTOS > 1_modulo
1  module.exports = {
2    soma(a, b) {
3      console.log(a + b)
4    },
5    multiplicacao(a,b){
6      console.log(a*b)
7    }
8  }
9
```

```
JS meu_modulo.js JS index.js X
curso_node-main > 2_FUNDAMENTOS > 1_modulo_interno > JS index.js
1  const meuModulo = require('./meu_modulo')
2  const soma = meuModulo.soma
3  const multiplicacao = meuModulo.multiplicacao
4  soma(2, 3)
5  multiplicacao(2,3)
6
7
```



Export e Import

- Com o Node.js também é possível utilizar o export e import do ES6;
- São funcionalidades mais modernas de importação e exportação;
- Com mais recursos do que as que vimos anteriormente;
- Para isso precisamos modificar os nossos arquivos para a extensão .mjs;
- E então podemos exportar uma com export default;
- E importar com import, uma única função, caso seja necessário;



Export e Import

- Exemplo Módulo usando import e export
 - Crie um arquivo meu_modulo.mjs e escreva uma função como no exemplo, depois crie um novo arquivo para o seu aplicativo principal com a extensão .mjs

```
JS meu_modulo.mjs X JS index.mjs
curso_node-main > 2_FUNDAMENTOS > 2_export_import > JS
1  export function soma(a, b) {
2    | console.log(a + b);
3  }
4
5  export function multiplicacao(a,b){
6    | console.log(a*b)
7  }
```

```
JS meu_modulo.mjs JS index.mjs X
curso_node-main > 2_FUNDAMENTOS > 2_export_import > JS index.mjs
1  import * as operacoes from "./meu_modulo.mjs";
2
3  operacoes.soma(2, 3);
4  operacoes.multiplicacao(2,3)
```


Core Modules

- No Node temos diversos Core Modules, que são os que vêm prontos para serem utilizados;
- Eles resolvem diversos problemas, como: trabalhar com arquivos e diretórios, servir aplicações e etc.
- Precisamos importar estes módulos no projeto para poder utilizar;
- Vamos utilizar um Core Module!



Core Modules

- Exemplo de Core Module – Ler arquivos com o módulo fs (FileSystem)
- Com o require (.js)
- Com o import (.mjs)

```
JS index.js ×
curso_node-main > 1_INTRO > 2_utilizando_modulo > JS index.js > ...
1  const fs = require('fs')
2
3  fs.readFile('arquivo.txt', 'utf8', (err, data) => {
4    console.log(data)
5  })
6
```

```
JS index.mjs ×  ≡ arquivo.txt
curso_node-main > 1_INTRO > 2_utilizando_modulo > JS index.mjs > ...
1  import fs from 'fs'
2
3  fs.readFile('arquivo.txt', 'utf8', (err, data) => {
4    console.log(data)
5  })
```



Core Modules

- Exemplo de Core Module – Ler arquivos com o módulo fs (FileSystem)
- Com o require (.js)
- Com o import (.mjs)

```
JS index.js ×
curso_node-main > 1_INTRO > 2_utilizando_modulo > JS index.js > ...
1  const fs = require('fs')
2
3  fs.readFile('arquivo.txt', 'utf8', (err, data) => {
4    console.log(data)
5  })
6
```

```
JS index.mjs ×  ≡ arquivo.txt
curso_node-main > 1_INTRO > 2_utilizando_modulo > JS index.mjs > ...
1  import fs from 'fs'
2
3  fs.readFile('arquivo.txt', 'utf8', (err, data) => {
4    console.log(data)
5  })
```



Ler argumentos

- O Node permite o envio de argumentos via linha de comando;
- Passamos eles após a instrução de execução do arquivo;
- Os argumentos ficam em um array chamado: `process.argv`
- Onde podemos fazer um loop e resgatar os valores enviados;
- Vamos ver na prática!



Ler argumentos

- Exemplo de uma forma de ler valores do terminal

JS index.js X

curso_node-main > 2_FUNDAMENTOS > 4_ler_argumentos > JS

```
1 console.log(process.argv);  
2  
3 const args = process.argv.slice(2);  
4 console.log(args);  
5  
6 const nome = args[0].split("=")[1];  
7 console.log(nome);  
8
```



Módulos externos

- Os módulos externos podem ser instalados via npm;
- Para isso precisamos inicializar o npm no projeto, com: `npm init`;
- A partir daí os módulos ficam mapeados e podemos instalar módulos;
- Que são salvos na pasta `node_modules`;
- Podemos instalar módulos com `npm install <nome>`;
- Vamos ver na prática



Módulos externos

- Vamos ver um exemplo da instalação e uso do módulo `minimist` para facilitar a leitura de dados a partir do terminal.

- Instalação do módulo

```
PS C:\Users\Leticia Lima\Desktop\curso_node-main\c
> npm install minimist
```

- Exemplo de utilização

```
JS index.js ...4_ler_argumentos JS index.js ...5_modulos_externos X
curso_node-main > 2_FUNDAMENTOS > 5_modulos_externos > JS index.js > ...
1 // --nome=Matheus --idade=30
2 const minimist = require("minimist");
3
4 const args = minimist(process.argv.slice(2));
5
6 console.log(args);
7
8 const nome = args["nome"];
9 const idade = args["idade"];
10
11 console.log(nome);
12 console.log(idade);
13
```

```
PS C:\Users\Leticia Lima\Desktop\curso_node-main\curso_node
ernos> node index.js --nome=Ana --idade=35
```



Algo prático com argumentos

- Podemos utilizar os argumentos recebidos para aplicar no nosso programa alguma lógica;
- Basta encapsular em variáveis e depois utilizá-los;
- Ou seja, podemos a partir do terminal, executar também uma função de um módulo interno nosso, por exemplo;



Aplicação prática

- Vamos ver um exemplo da utilização de valores coletados no terminal para utilizar no index.js

```
JS index.js X
curso_node-main > 2_FUNDAMENTOS > 6_pratica_com_args > JS index.js > ..
1  const minimist = require("minimist");
2  const meuModulo = require("./meu_modulo");
3
4  const soma = meuModulo.soma;
5  const multiplicacao = meuModulo.multiplicacao;
6
7  const args = minimist(process.argv.slice(2));
8
9  const x = args["x"];
10 const y = args["y"];
11
12 soma(x, y);
13 multiplicacao(x,y)
14
```

```
JS meu_modulo.js X
curso_node-main > 2_FUNDAMENTOS > 6_pratica_co
1  module.exports = {
2    soma(a, b) {
3      console.log(a + b)
4    },
5    multiplicacao(a,b){
6      console.log(a*b)
7    }
8  }
9
```



Melhorando a visualização

- Há um módulo externo chamado chalk;
- Ele pode deixar a visualização do console mais agradável;
- Fazendo com que seja possível expressar um feedback com base em cores;
- Vamos ver na prática!



Melhorando a visualização

- Exemplo: Instalação e utilização do chalk
- Exemplo: instalação do chalk

```
PS C:\Users\Leticia Lima\Desktop\curso_node-main\cu  
_args> npm install chalk
```

- Exemplo: Utilização

```
JS index.js  X  
curso_node-main > 2_FUNDAMENTOS > 8_melhorando_visual > JS index.js > ...  
1  const chalk = require("chalk");  
2  const nota = 8;  
3  if (nota >= 7) {  
4    console.log(chalk.green.bold("Parabéns, você passou!"));  
5  } else {  
6    console.log(chalk.bgRed.black("Você precisa fazer a prova final!"));  
7  }
```



Lendo entrada de dados

- Podemos ler dados do usuário com o módulo readline, um Core Module;
- Neste caso utilizamos o método question, que faz uma pergunta a ser respondida pelo usuário;
- Depois podemos processar a resposta e entregar um retorno;
- Vamos ver na prática!



Lendo entrada de dados

- Exemplo de como utilizar

JS index.js X

curso_node-main > 2_FUNDAMENTOS > 9_lendo_input > JS index.js > ...

```
1  const readline = require("readline").createInterface({
2    input: process.stdin,
3    output: process.stdout,
4  });
5
6  readline.question(`Qual a sua linguagem preferida? `, (language) => {
7    console.log(`A minha linguagem preferida é: ${language}`);
8    readline.close();
9  });
10
```



Melhorando a leitura de dados

- Há um módulo externo chamado `inquirer`;
- Que é muito mais completo para resgatar e lidar com o input do usuário;
- Além disso, é baseado em Promises, o que torna sua utilização mais simples;
- Vamos ver na prática!



Melhorando a leitura de dados

- Exemplo de utilização

```
npm install inquirer
```

```
JS index.js ×
curso_node-main > 2_FUNDAMENTOS > 10_abstracao_input > JS index.js > ...
1  const inquirer = require('inquirer')
2  inquirer
3    .prompt([
4    { name: 'p1', message: 'Qual a primeira nota?' },
5    { name: 'p2', message: 'Qual a segunda nota?' },
6    ])
7    .then((resp) => {
8      console.log(resp)
9      const media = (parseInt(resp.p1) + parseInt(resp.p2)) / 2
10
11      console.log(`A média do aluno é ${media}`)
12    })
13    .catch((err) => {
14      console.log(err)
15    })
```



Erros no Node

- Temos duas formas principais para gerar ou evidenciar erros em Node.js;
 - throw: uma forma de encerrar um programa, gerando um novo erro;
 - try catch: uma forma de evidenciar algo que deu errado em um bloco de código e exibir a mensagem de erro;
 - Vamos ver na prática!



Erros no Node

- Exemplo throw

JS throw.js ✕

curso_node-main > 2_FUNDAMENTOS > 14_erros > JS throw.js > ...

```
1  const x = "10";
2
3  if (!Number.isInteger(x)) {
4    throw new Error("O valor de x não é um número inteiro");
5  }
6
```

- Exemplo try e catch

JS try_catch.js ✕

curso_node-main > 2_FUNDAMENTOS > 14_erros > JS try_catch.js > ...

```
1  const x = 10;
2
3  try {
4    x = 2;
5  } catch (err) {
6    console.log(`Erro: ${err}`);
7  }
8
```



Tarefa 01

1. Crie um novo projeto de Node.js;
2. Crie um arquivo para a aplicação com o nome programa;
3. No arquivo crie duas variáveis e imprima a soma delas;
4. Execute o arquivo e verifique a resposta no terminal;




Tarefa 02

1. Crie um novo projeto que aceite pacotes externos;
2. Instale o inquirer e o chalk;
3. Utilize o inquirer para receber o nome e a idade do usuário;
4. Apresente esta resposta com uma cor de fundo amarela e texto preto;
5. Dica: Você pode utilizar `bgYellow` e `black`!
6. Insira um tratamento para um possível erro do inquirer com o `catch`;





(85) 98524-9935  youthspace

 contato@youthidiomas.com.br

<https://www.youthspace.com.br/>

2023

